

GPU-based Parallelization for Schedule Optimization with Uncertainty

Christabelle Bosson* and Min Xue†

University of California at Santa Cruz, Moffett Field, CA 94035

Shannon Zelinski‡

NASA Ames Research Center, Moffett Field, CA 94035

This paper presents an application of Graphics Processing Units (GPU) technology for speeding up a schedule optimization problem under uncertainty and provides a fast decision support algorithm to solve an air traffic management problem. In terminal airspace, integrated departure and arrival operations using shared resources have the potential to increase operations efficiency. However, results and benefits from integrated operations might be sensitive to flight time uncertainty. In previous work, a scheduling algorithm was proposed for a model of the Los Angeles terminal airspace. Uncertainty was introduced in the flight times and the uncertainty cost computation was handled by Monte Carlo simulations. The original implementation was carried out on sequential processors, but a 30-minute scenario ran in 6.5 hours, which prohibits applying the algorithm in real-time. This paper presents a GPU-based implementation of the scheduling optimization with uncertainty achieving a 637x speedup in Monte Carlo simulations and a 154x speedup for the entire algorithm compared to a sequential implementation. The runtime of the GPU-based code for the same 30-minute scenario is about 2.5 minutes. This significant speedup allows a large range of experiments to be explored and hundreds of simulations to be run. Two types of experiments are designed and they explore different values of traffic densities and arrival-to-departure ratios. The results demonstrate that there exist trade-off solutions between computed delays and number of controller interventions. The variation of total number of aircraft showed a larger impact on the controller's workload than the variation of arrival-to-departure ratios. When the traffic density is increased, compromise solutions can be identified to reduce the number of controller interventions and achieve low delays.

I. Introduction

Terminal airspaces are characterized by narrow portions of airspace in which high traffic volumes must fly through in short time periods. In these constrained environments, most aircraft are climbing or descending at various speeds. In dense terminal operation areas in which busy airports are close to each other, severe bottlenecks can be formed affecting the National Airspace System (NAS) efficiency. Therefore there is a clear need to facilitate air traffic operations and improve their efficiency in the terminal airspace.

Over the past few decades, many research efforts attempted to improve air traffic operations in terminal areas by solving arrival scheduling problems,¹⁻⁶ departure scheduling problems,⁷⁻⁹ surface management problems^{10,11} and integrated arrivals and departures problems.¹²⁻¹⁶ Recent studies^{12,17-22} showed that integrated departures and arrivals have the ability to improve the operations efficiency when active metroplex areas are nearby. However, benefits obtained and flight schedules computed under deterministic conditions in terminal operations areas are subject to uncertainty. Uncertainty comes from many sources, such as human factors, aircraft dynamics, wind prediction or weather forecasts that are difficult to model accurately.

Uncertainty analyses were conducted to help estimate the robustness of the solutions and benefits obtained from integrated operations. In arrival scheduling problems, Thipphavong et al. used the Stochastic Terminal Arrival Scheduling Software (STASS) to study the relationship between uncertainty and system performance²³. For the integrated

*PhD Student, University Affiliated Research Center. AIAA Student Member

†Research Scientist, University Affiliated Research Center. Mail Stop 210-8. AIAA senior member

‡Computer Engineer, Aerospace High Density Operations Branch. Mail Stop 210-6. AIAA senior member

departures and arrivals problem, Xue et al. analysed the consequences of flight time uncertainty on the integrated scheduled operations by investigating the impacts on delays and controller workloads¹⁴.

The integration of uncertainty in algorithm formulations is crucial to better reflect the reality of current air traffic operations. One way to accommodate uncertainty in algorithms is to use buffering or probabilistic sampling techniques. In scheduling problems, the propagation of uncertainty can be represented by Monte Carlo simulations. In previous work, Xue et al. used sequential processors to develop a scheduling algorithm for integrated arrival and departure operations on a model of the Los Angeles terminal airspace¹³. Uncertainty was introduced in the flight times and handled by Monte Carlo simulations. To run Monte Carlo simulations, thousands of sampling points were used. However, solving large scale stochastic optimization problems can become computationally challenging, requiring many hours of computation, which makes testing and application prohibitive.

General purpose applications that are computationally expensive are benefiting from the emergence of programmable Graphics Processor Units (GPUs). NVIDIA released a parallel computing development platform called Compute Unified Device Architecture (CUDA)²⁴ that is implemented by GPUs produced by the company. The programming model uses an environment based on the C/C++ language and existing libraries²⁵. CUDA allows developers to use GPUs for general purpose applications that are not exclusively graphics, and are suitable for high performance computing applications^{26,27} due to their parallel throughput architecture. Many concurrent threads can be launched at the same time to process individual instructions over multiple data (SIMD). In addition, they represent an investment at low cost when compared to multiple CPU cores. In terms of applications, many areas have already shown significant advantages of using GPUs illustrated with important computing performance increases, for example multitasking²⁸, medical applications^{29,30} or finance^{31,32}. For air traffic management applications, Tandale et al. provided one of the first GPU study contribution and accelerated by 30 times a CPU implementation of a large-scale Traffic Flow Management (TFM) problem with 17,000 aircraft³³. In terms of algorithms, evolutionary computing techniques present an inherent parallel nature whose performance can be significantly improved when implemented on GPUs. Recent studies^{34–39} show important computation time savings and large speedup when genetic algorithms are run on GPUs. The GPU's current maturity and its great capability allow a large community of researchers to successfully solve complex problems by exploiting such technology.

This paper contributes to the air traffic management field by providing a fast decision support algorithm enabled by the use of GPU. It provides a GPU-based parallelization of an existent scheduling optimization¹³ with uncertainty. The Monte Carlo simulations used for the uncertainty cost computation are transferred to the GPU in order to save computation time. Parallelization strategies are investigated and applied to data transfer, random number generation and GPU-occupancy optimization. Overall, the performance of the algorithm was significantly increased. The resulting acceleration enables a large range of experiments to be explored and hundreds of scenarios to be rapidly tested. An analysis is carried out on two types of experiments which explore different values of traffic densities and arrival-to-departure ratios. The characterization of the algorithm and solutions obtained are also provided as well as the extraction of delay and controller intervention results and trends.

In the paper, the sections are organized as follows. Section II provides some background on the problem studied and revisits briefly the existing algorithm implementation. The GPU implementation of the scheduler is described in Section III. Different accelerating strategies and associated performances are also provided in Section III. An application of the GPU-based scheduler in the Los Angeles terminal airspace is presented in Section IV. Section V presents multiple scenario evaluations and corresponding results characterization. Finally, conclusions and future research are provided in Section VI.

II. Background

This section provides background on the existing model of optimization with uncertainty, its original implementation and its application.

A. Los Angeles case study

This effort is based on previous work¹³ studying optimal integrated operations with uncertainty for partial flows in the Los Angeles terminal airspace. The interactions between arrivals and departures in this area are worth a special case study because of their complex natures and layouts.

Figure 1 shows the Standard Terminal Arrival Routes (STARs) and the Standard Instrumental Departures (SIDs)



Figure 1: Route interactions between arrivals and departures in the LA terminal airspace

procedures that aircraft need to follow when flying within the Los Angeles terminal airspace. The arrival procedure (STAR), SADDE6, is associated with arrivals via the FIM fix (waypoint on Figure 1) and requires that arrivals follow the route FIM-SYMON-SADDE-SMO to land at the Los Angeles international airport (LAX). The departure procedure (SID), CASTA2, is associated with departures to the North from Runway 24L and requires departures to follow the route RWY-NAANC-GHART-SYLEX. In this paper, SADDE6 and CASTA2 are denoted indirect routes. The figure demonstrates that multiple routes could be flown for both of these procedures. A direct route for departures would be RWY-WPT2-WPT1 and a direct route for arrivals would be FIM-WPT1-SMO where WPT1 and WPT2 are fixes made up to simplify the route descriptions. However, current operations impose altitude constraints at fix GHART. Arrival flights have to maintain their altitude above 12,000 feet (FL120) at fix GHART and departure flights have to fly at or below 9,000 feet (FL90) at the same fix. Due to the restrictions at GHART, both arrivals and departures are routed to WPT1 and WPT2 which increase the total flight time up to 2 minutes. In current operations, Timar et al. found that 28.1% of LAX arrivals follow the SADDE6 procedure and 10.4% of LAX departures follow the CASTA2 procedure.⁴⁰ On average, this can be translated to 220 arrivals and 80 departures per day or 9 arrivals and 3 departures per hour. The model considered cover partial flows of the Los Angeles terminal airspace.

B. Optimization with uncertainty

The optimization model is formulated as a multiple objective optimization with both spatial and temporal separations as aircraft separation methods. The objective is to minimize the total amount of delay for all departure and arrival flights and the number of controller interventions. In the formulation, the delay is equivalent to the sum of exit times at waypoint SUTIE for arrivals and at waypoint WPT1 for departures. A controller intervention is defined as the need to intervene to avoid a loss of separation between aircraft. Deterministic solutions are first computed using a non-dominated sorting genetic algorithm (NSGA). Because flight times are subject to uncertainty, unexpected separation loss or flight delay might occur. Therefore, error sources are added to the flight times in the model to generate uncertainties. Work performed on trajectory and flight time predictions show that predicted flight delays can be represented by shifted normal distributions.^{41,42} Therefore as illustrated in Figure 2, samples drawn from a normal distribution are used for simplicity to perturb flight times at FIM and SUTIE for arrival flows and at the runway for departure flights¹⁴. When separation losses occur, Monte Carlo simulations are used to simulate controller interventions. The design variables allow planes to fly two route options (direct or indirect), fly at different speeds and allow both air and ground delays to be computed to ensure conflict-free operations. The route assignments are determined by the solutions generated under deterministic scenarios and are not modified by the Monte Carlo simulations. The controllers

are assumed to follow a First-Come-First-Serve (FCFS) rule to process the aircraft. The separation requirements at the fixes are hard constraints. If these are not respected, extra delay will be imposed which will involve extra controller interventions. More details about the optimization formulation and handling of uncertainty are covered in previous work¹³.

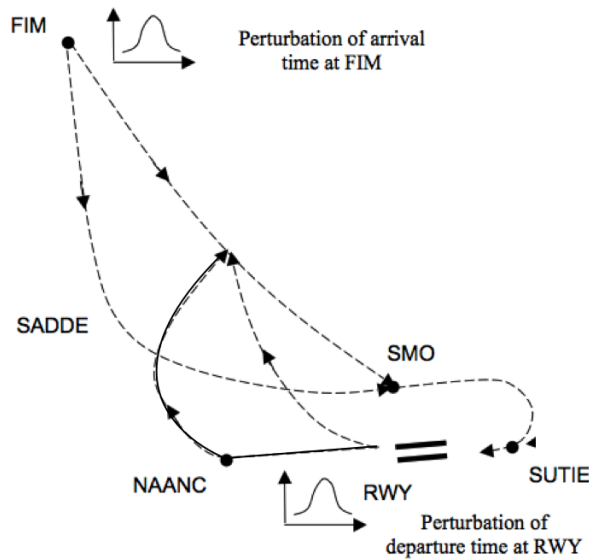


Figure 2: Flight time perturbations setup

C. Existing CPU implementation

The original implementation of the stochastic scheduler reported in previous work¹³ was carried out on traditional sequential processors. Because genetic algorithms (GAs) are sensitive to the initial population, different seeds corresponding to different starting populations were explored. To compute the various seeds at the same time, multi-threading was coded using Tcl scripting to parallelize the computation on the CPU and distribute the jobs evenly among the CPU cores. The whole procedure is illustrated in Figure 3.

Figure 3 shows that each thread is assigned to run on different seeds. The structure of the code allows each thread to run smaller operation chunks. The code uses 10 seeds which are distributed among 4 Tcl threads. In the GA, the population size is set to 832 individuals. The cost evaluation function consists of two different steps, the deterministic cost evaluation and the uncertainty cost computation. The cost function evaluation is the most time consuming process because it computes the uncertainty cost for each individual, which is itself evaluated by thousands of Monte Carlo simulations.

III. GPU Implementation

This section presents and describes the implementation of the stochastic scheduler on a mixed CPU-GPU (host-device) platform development environment. Accelerating strategies are investigated to optimize the overall performance. The results that are presented in the following sub-sections are obtained using the author's implementation. The implementation is novel and was not used in previous work.

In this work, the NVIDIA GeForce GTX690 was used on a Linux platform with 2.30 GHz 12-Core Intel Xeon and 16 GB RAM. The graphics card has a CUDA Capability of 3.0 and combines two Kepler GPUs, consisting each of 1,536 CUDA Cores. The NVIDIA GeForce GTX690 has a total global memory of 2,048 MB and a total shared memory of 49,152 bytes per block. The maximum number of threads per streaming processor is 2,048 and the number

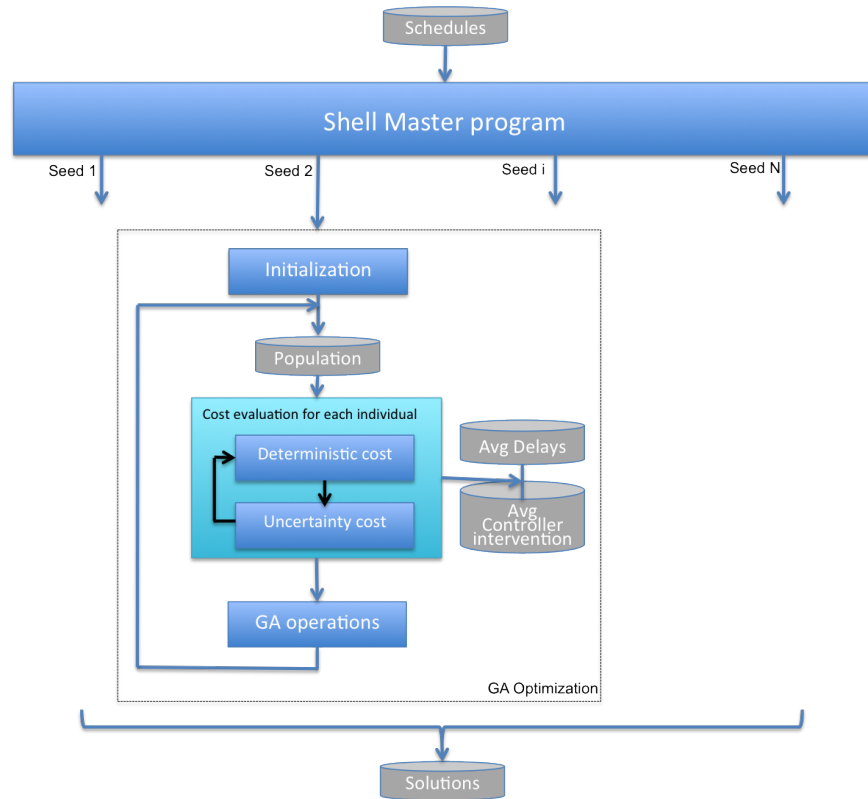


Figure 3: Integrated arrivals and departures scheduler with uncertainty - Original Implementation¹³

of threads per block cannot exceed 1,024. A maximum of 16 blocks per streaming processor can be launched in parallel.

A. Implementation of the Monte Carlo simulations on the GPU

The robustness of the uncertainty cost computation depends directly on the number of Monte Carlo simulations evaluated. When a large number of Monte Carlo simulations is computed on the CPU, it becomes a computational challenge and compromises the optimization feasibility. Monte Carlo simulations rely on a large number of independent simulations which make them perfectly suitable for parallelization. A GPU is therefore introduced to offload thousands of uncertainty evaluations from the CPU and reduce the associated computation time.

The following pseudo-algorithm provides the processing model of the parallel implementation of the Monte Carlo simulations on the GPU. Each Monte Carlo simulation requires a large amount of intermediate variables to evaluate the two objectives of this optimization. Therefore the global memory of the GPU is used to store them.

Algorithm 1: A single Monte Carlo simulation evaluating the uncertainty cost for a set of schedule-scenarios

```

1: procedure MONTE CARLO(mean, standardDeviation, delay, workloadCounter)
2:   for all schedules  $\in$  setof(departure, arrival) flights do                                ▷ computation on CPU
3:     Generate random number from given normal distribution ▷ create uncertainties on departure/arrival times
4:   end for
5:   for MonteCarloSimu = 0  $\rightarrow$  maxNumberMonteCarloSim do                                ▷ computation on GPU
6:     Delay and intervention function evaluation                                       ▷ 1 MC sim. per thread
       return Delay and interventionCounter
7:   end for
8:   for all individual  $\in$  (GApopulation) do                                       ▷ computation on CPU

```

```

9:         Average delay and workloadCounter over maxNumberMonteCarloSim
10:    end for
11: end procedure

```

Thousands of Monte Carlo simulations are computed in parallel for each individual. In CUDA terms, this is performed by launching from the host at the same time 32 thread blocks, each containing 128 threads.

The GPU-based implementation of the Monte Carlo simulations consists of a kernel that is called from the host. For each individual of the GA population, the kernel computes thousands of uncertainty costs. Because the Monte Carlo simulations of each seed are grouped together, millions of operations are performed simultaneously on the GPU. In order to enable the computation of these Monte Carlo simulations on the GPU, a combination of block number and thread number per block is assigned. Because the number of blocks that can be called and run simultaneously is limited by the physical properties of the graphics card used, the entire population cannot be processed at the same time on the GPU. Therefore the population is split into smaller group chunks, and a loop is created to handle all the chunks. The degree of parallelism is not explicitly expressed on the GPU. The kernels look like serial programs, and functions are written as if they would be run by a single thread. To address this, different thread granularity options were investigated. By launching a large number of threads and a large number of thread blocks, the GPU runs the program in parallel on many threads (128) and many blocks (32).

A high-level flow chart is provided in Figure 4 and represents the task organization and mechanism of the scheduler. In this current implementation, 10 seeds are distributed among 10 CPU-cores. The threads synchronize before the GPU call so that the whole population set is sent to the GPU. Figure 4 shows that the inner loop on each individual originally represented in Figure 3 has been removed for efficiency.

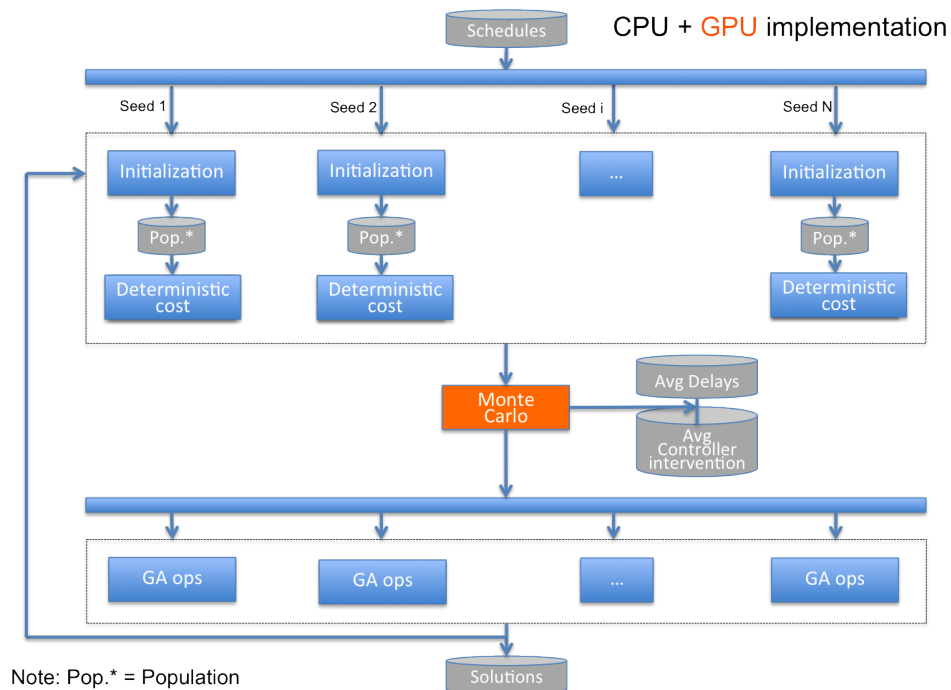


Figure 4: Integrated arrivals and departures scheduler with uncertainty - Final implementation in current study

B. Accelerating strategies

Accelerating strategies are investigated for the purpose of increasing speed. The parallelization strategy chosen induces the distribution of problem instances to multiple GPU-processors. A code performance evaluation study is conducted in order to show different tuning strategies that can be used to improve the performance.

1. Data transfer

Because the host-device communication between CPU and GPU is time consuming, the operations of memory allocation and data transfer have to be minimized. For the GPU-based implementation of the Monte Carlo simulations, data needed in the GPU-kernel are first copied from the host memory to the device memory. Then, necessary memory is allocated on the device. This data-related operation requires two CUDA function calls: CUDA malloc and CUDA memcpy.

This test compares two alternative strategies to perform the memory allocation and data transfer operations. On one hand, the first method allocates and copies the input data separately for each individual. Therefore this first method requires 832 CUDA malloc and 832 CUDA memcpy to address the whole population. On the other hand, in the second method, the input data is allocated and copied only once for the whole population. Only one CUDA malloc and one CUDA memcpy are then required.

Table 1 presents the resulting computation times for both strategies when one generation is considered in the simulation. The computation times obtained show that calling CUDA memory functions multiple times increases the computation time of the Monte Carlo simulations by 81.8x.

Table 1: Effect of CUDA function call grouping on Monte Carlo computation times

Number of CUDA malloc and CUDA memcpy	Size unit	Monte Carlo computation time using GPU for one generation
832 CUDA malloc and 832 CUDA memcpy	1 individual	26.75 secs
1 CUDA malloc and 1 CUDA memcpy	832 individuals	0.327 secs

This test shows that host-device communication through the use of CUDA function calls can be very expensive when used intensively. Therefore, the code performance can be improved when memory operations are executed in a small number. The flow chart of the original code is modified to address this and is presented in Figure 5. The uncertainty cost evaluation loop over each individual has been replaced by a single large memory operation on the GPU. Once the deterministic cost is evaluated for a whole population on the CPU, the results for the population are transferred to the GPU and the uncertainty cost is computed.

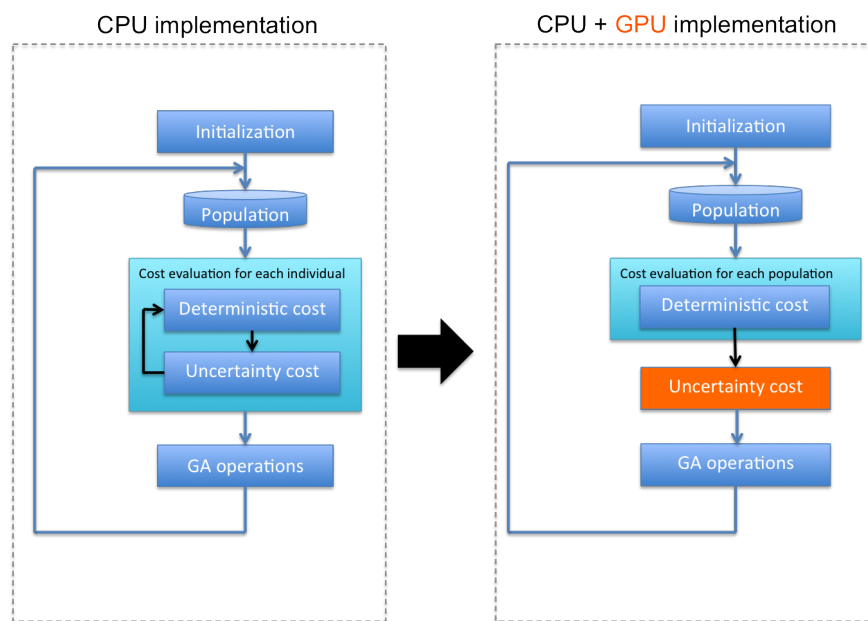


Figure 5: Modification of the original flow chart (see Figure 3), resulting in less host-device communication

2. Random numbers generation

In the Monte Carlo evaluation, random numbers that will be added as source errors in the flight times have to be generated for each simulation. Random numbers are required to follow a normal distribution. Therefore, this section explores different ways of generating random numbers following a normal distribution on the CPU and on the GPU. Five generators are implemented, two on the GPU and three on the CPU. In this algorithm, 1,024 Monte Carlo simulations are evaluated per population. In the reference scenario 14 flights are considered, therefore $14 \times 1,024 = 14,336$ random variables are needed per population. The aim of this test is to compare running times of the different random number generators. Each random number generation is performed in a separate function or kernel. Resulting random numbers are gathered in a main CPU function which also computes the different running times. To make sure that overheads and warmup times are minimized in the computation times for all benchmarks, 100 generation trials are performed and resulting running times are averaged. The different types of random number generators and associated averaged running times can be viewed in Table 2.

Table 2: Comparison of different random number generators

Generator	Type	Location	Average running time
1	cuRAND library, MRG32k3a	GPU	1.97 ms
2	cuRAND library, XORWOW	GPU	2.08 ms
3	Boost library, Mersenne 19937	CPU	2.65 ms
4	Boost library, Ecuyer 1988	CPU	2.60 ms
5	C-function Gaussian	CPU	1.2 ms

This test shows that GPU might not be very efficient at generating large sets of random numbers. Moreover, this test demonstrates that using the Boost library on the CPU is more time consuming than implementing a C-function. Generator 5 was implemented by hand and uses the Box-Muller transformation⁴³ to transform uniformly distributed random variables to a new set of random variables with a normal distribution. Based on the average running times, this test demonstrates that Generator 5 is more capable for this task and is therefore chosen for this implementation.

3. GPU-occupancy study

The Monte Carlo simulations require thousands of evaluations to cover each individual. The hardware being used has physical limitations and cannot compute every single evaluation simultaneously. Thus the Monte Carlo kernel has to be launched multiple times to ensure the completion of the computation. In CUDA terms, a grid size defined by a fixed number of blocks and a fixed number of threads per block has to be chosen. To ensure that the grid configuration chosen is optimal, an occupancy study is performed using the CUDA Occupancy calculator tool provided by NVIDIA.

The GPU-occupancy is defined as the number of active thread blocks per processor divided by the maximum number of thread blocks per processor. It is a number in the range of 0 to 100%. At any time, some threads may be processing data and some others may be accessing memory. When this occurs, the number of threads available for computation is limited, degrading the occupancy. Occupancy of 100% means that the GPU-cores are fully utilized. The internal GPU thread block scheduler controls when thread blocks are executing data. Therefore, understanding how the scheduler works is crucial. According to Luitjens and Rennich⁴⁴, the occupancy calculator can help efficiently use the GPU. However, the resulting configurations of the occupancy study might not provide optimal computation times.

To reach an optimal trade-off between occupancy and computation time, a few experiments are conducted to investigate different grid configurations for the Monte Carlo kernel. The inputs needed by the calculator are GPU-related (compute capability and size of total shared memory) and kernel-specific (grid size, number of registers and size of shared memory used). The amount of registers and shared memory provided to the tool are defined by how the kernel is coded. The Monte Carlo kernel implemented requires 32 registers and 167 bytes of shared memory. The outputs of the calculator are the resulting occupancy and occupancy limiters. Occupancy limiters are given so that the user gets an idea of which parameter (grid, register or shared memory) constrain the number of threads per block that can be launched. Three tests are conducted to study how the grid size affects the GPU-occupancy.

Table 3 gathers occupancy results when the grid size and the number of threads per block are varied. Because the total number of threads is capped at 2,048 for the GPU used in this work, the number of active blocks will decrease

Table 3: GPU-occupancy study

# threads	Active thread blocks	Occupancy
64	16	50%
128	16	100%
256	8	100%
512	4	100%

when the number of threads per block increases. If higher numbers of blocks are defined than the one defined by the GPU physical limits, they will be active sequentially. The block sequencing is arranged internally by the GPU scheduler.

The outcome of the occupancy study shows that high GPU-occupancy is obtained when the number of threads launched at the same time is large, or in other words, occupancy is high when the number of Monte Carlo simulations computed in parallel is large. Maximum occupancy is reached for the last three rows corresponding to three different numbers of threads per block. To find the optimal trade-off between GPU-occupancy and computation time, the code is run for the last three rows. Table 4 presents the GPU-kernel and overall algorithm computation times associated when one generation is simulated. Despite 100% occupancy obtained for block sizes of 256 or 512 threads, the corresponding running times are slower. From these results, the solution that compromises occupancy and computation time is obtained when each block contains 128 threads.

Table 4: GPU and CPU Computation times for one generation using different number of threads

# threads	GPU-kernel Computation Time	Algorithm Computation Time
128	0.031 secs	0.382 sec
256	0.032 secs	0.388 sec
512	0.034 secs	0.391 sec

The following set of plots in Figure 6 illustrates how the inputs of the CUDA occupancy calculator can influence and impact the occupancy of the GPU. The curves illustrate how the different input impact the occupancy when the inputs values are varied. The triangle shows the occupancy of the Monte Carlo kernel when the number of threads per block is set to 128. Because the registers and shared memory usages are fixed by the kernel, the top graph shows that for this kernel a block size of 128 threads is the optimal setting in terms of computation time. In addition, the curves show that maximum occupancy can be obtained for different numbers of threads per block, such as 256 or 512 as previously tested. The middle and bottom graphs provide a visual representation of how occupancy can be degraded and demonstrate that memory coding choices might affect the GPU-occupancy.

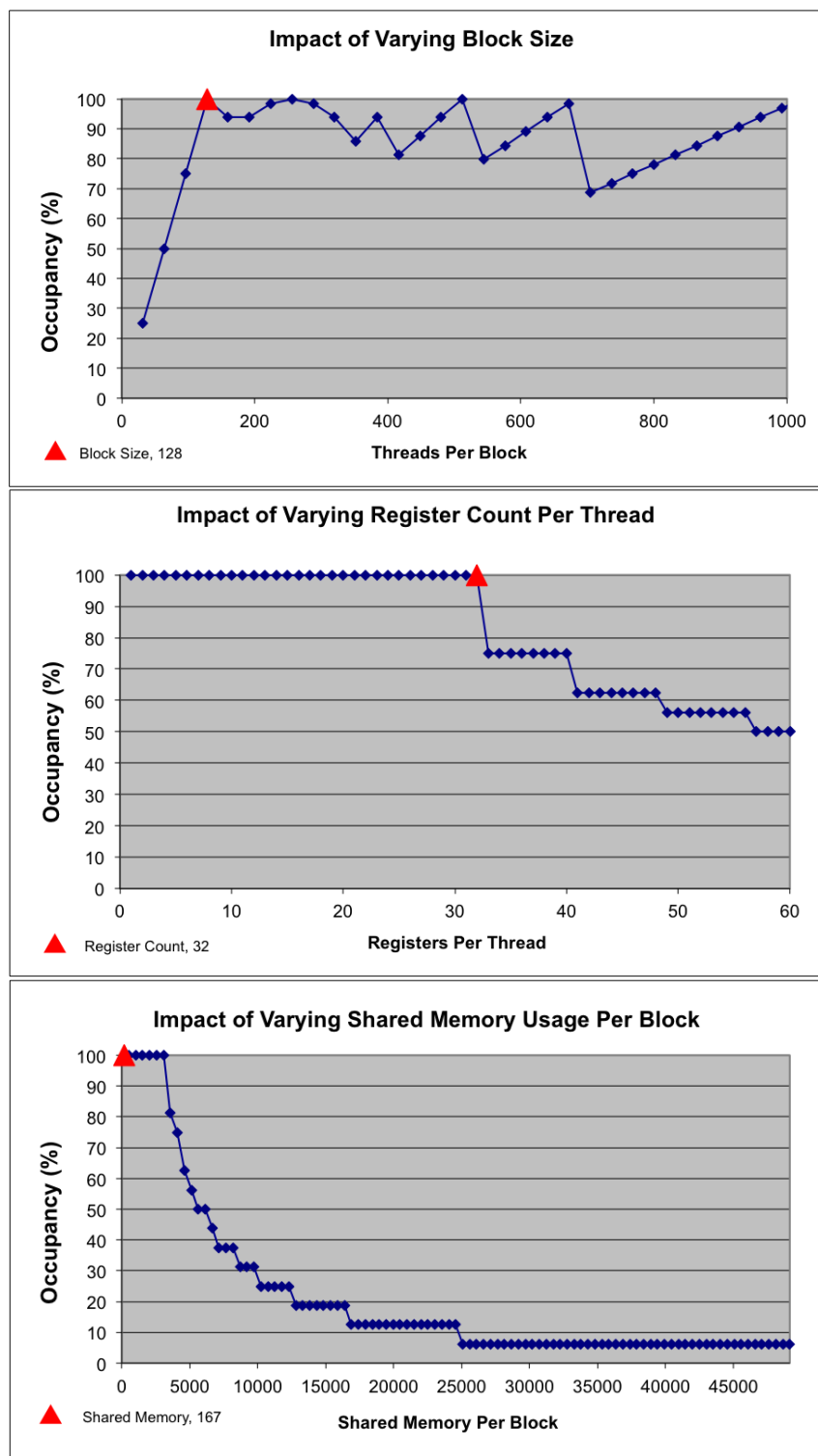
C. Performance results

The different experiments provide some interesting results, some of which are unexpected. To illustrate the performance achieved by the final version of the code, the algorithm runs 400 generations. The number of Monte Carlo simulations run for each individual is 1,024 and 10 seeds are considered. Table 5 presents the comparison of the computation times between the original and the current code.

Table 5: Comparison of computation times between original and current version of the code

Code compared	# seeds	# generations	Original code	Current code using 2 GPUs	Speedup
Monte Carlo simulations	1	1	18.5 secs	0.029 secs	637x
Total	10	400	23,400 secs (6.5h)	152.19 secs	154x

The GPU computes 1,024 Monte Carlo simulations for $popSize * numSeeds = 8,320$ individuals. Thanks to the GPU, the computation time of the Monte Carlo simulations has been decreased from 18.5 seconds to 0.029 seconds. The GPU performs well at 8.5 million operations and significantly outperforms the original inner loop computing only 1,024 operations at a time. The computation time is now 152.19 seconds, which is less than 2.5 minutes. The code



IV. Application to the Los Angeles terminal airspace

This section provides a direct application of the scheduling tool supported by GPU technology on a representative schedule of 14 flights flying on partial flows in the Los Angeles terminal airspace. The schedule covers 30 minutes of historical traffic operations on December 4, 2012 from 9:00 AM to 9:30 AM. It includes two flows: 8 arrivals from FIM fix and 6 departures to the North from Runway 24L. According to statistical data mentioned previously, the numbers of flights considered in the arrival flow represent typical operational traffic on SADDE6. However the number of flights considered in the departure flow is twice the average number of departures using CASTA2. Therefore, because the aircraft will fly in a mixed environment with shared resources, 14 flights represent a typical scenario case that requires scheduling. Table 6 provides a representative schedule of the 14 flights considered. The initial times are relative to simulation start time and the flights order is determined through a sorted ordering of the initial times.

Table 6: Scheduled initial times

Flight Order	FIM Arrivals (sec)	RWY Departures (sec)
1	39	68
2	446	165
3	728	363
4	1106	529
5	1332	1613
6	1475	1830
7	1613	NA
8	1770	NA

In previous work¹³, Xue et al. applied the scheduling tool developed on sequential processors to this representative schedule and presented the solutions obtained that were optimized under uncertainty. The results showed that under uncertain environment, the integrated arrivals and departures in terminal airspace can provide great savings in flight delays. Moreover, the proposed method demonstrated trade-offs between delay savings and controller intervention counts. However, the optimization process took about 6.5 hours which would be a problem for real-time application and was prohibitive for further experiments and extension of the algorithm.

In this present work, the GPU-based scheduling optimization was applied to the same scenario. The solutions obtained and results trends are the same as with the sequential version of the scheduling tool. However, the GPU technique enabled solving the optimization process in about 2.5 minutes resulting in a speedup of 154x. With this significant speedup, the GPU-based scheduling tool now presents characteristics to be a candidate for real-time/fast-time tool.

V. Experiments and characterization of the algorithm

Previous section showed that the application of the GPU-based scheduling tool to a concrete traffic flow scenario presents great benefits in terms of computing time. The important speedup obtained allows hundreds of scenarios to be run quickly. The main aim of this section is to use the speedup gain brought by the GPU technique to test diverse experiments. This section describes the experiments that were designed along with the results obtained. The goal of these experiments is to characterize the algorithm and understand its performance on a large range of different scenarios. Scenarios are generated randomly but the computed schedules are restrained to respect the authorized aircraft separation times. Some scenarios might not necessarily represent realistic conditions as they might stress the algorithm outside its solution space search. The representative scenario tested in previous section establishes the baseline. Two different types of experiments are designed and tested. Both types investigate the variation of a single parameter that can take different values on a defined range. Details are provided in Table 7. For each specific value of the defined range of the total number of aircraft, 400 schedule-scenarios are randomly generated then tested in simulations.

In the first experiment, the range chosen for the total number of aircraft starts with 14 aircraft which represents average traffic operations for the partial flows considered in the Los Angeles terminal airspace. The total number of aircraft to be scheduled is then increased up to 24 aircraft. One of the goals of the experiment is to test the algorithm

Table 7: Experiments setup

Experiment	Varied parameter	Range of values tested
1	total number of aircraft	14 to 24
2	arrival-to-departure ratio	0.05 to 8.00

with higher than normal (70% greater than typical) traffic operations. For each number of aircraft, a random number of arrivals is computed. For example, if the number of aircraft is fixed to 18, the number of arrivals can take any integer value between one and 18. Because 400 scenarios are generated per value of the number of aircraft, up to 400 different arrival-to-departure ratios can be generated. The lower fixed ratio is 0.05 and represents almost only departures (e.g. one arrival and 20 departures) whereas the maximum fixed ratio is 8 and represents mainly arrivals (e.g. 16 arrivals and two departures). Based on average traffic data for the partial flows considered in the Los Angeles terminal airspace, any ratio equal to $3 \pm 20\%$ represents typical operations.

For each schedule-scenario, new initial flight times are generated as pseudo-random numbers that follow a uniform distribution. It is assumed that the probability of having a flight time in a 1-minute interval is the same as having another flight time in any 1-minute interval but the time values are constrained to stay within a 30-minute time range. A uniform distribution was used to allow more varied schedule permutations. Because the minimum separation distance is four nautical miles, and the speed range considered in the model implementation is between 180 kts and 350 kts, aircraft separation times are maintained between 41 and 80 seconds. In this work, the aircraft fleet considered is homogeneous flights are assumed to be the same aircraft type.

For analysis purposes, a set of Pareto fronts are computed for each schedule-scenario. Because ten different seeds are considered, the algorithm computes ten different fronts per scenario. Previous Pareto front studies focusing on multi-objective optimization^{45,46} showed that the leading edge is a good representation of the optimal front. Therefore in this application, the leading edge of the 400 Pareto fronts is retrieved for each simulation to provide a visual representation of the set of minimum pairs (controller intervention and delay).

A. Experiment 1: Variation of the total number of aircraft

1. Setup

In the first experiment, ten different values of the total number of aircraft are tested; the range of values considered is provided in Table 7. For each value of the total number of aircraft, a simulation is run to test 400 schedule-scenarios. Although the total number of aircraft is kept as a constant in the scenario generation procedure, the number of arrivals and the number of departures are randomly computed while constraining their sum. Therefore the arrival-to-departure ratio is also a random variable.

2. Results

Figures 7-8-9 present the superposed results of each simulation. As the number of total aircraft was increased, the computation time of each simulation increased. To run 400 scenarios, the computation time varied between 5 hours when the number of aircraft was fixed to 24 and a maximum of 20 hours when the number of aircraft was fixed to 14. This large computation time variation illustrates that the program cannot compute solutions when testing scenarios with 40% greater than typical number of aircraft.

Because the algorithm has a fixed delay range that can be assigned, it cannot assign delays outside this range. Therefore if the program encounters this situation, it cannot compute a solution. Define an infeasible scenario as a scenario for which the algorithm cannot compute a solution. After each simulation, data were analyzed to determine the number of scenarios for which the algorithm could not find a solution (i.e. infeasible). Once infeasible scenarios were discarded from the solution set, a time discretization was performed on the range of delay obtained. Then, the number of scenarios having a solution in each time delay bin was computed. Figure 7 shows the scenario distributions of each simulation.

After each simulation, the leading edge of the 400 Pareto fronts was retrieved and saved. When every simulation was done, the global leading edge was computed from the ten different leading edges. The global delay range was sampled with a time delay bin size of five seconds and the averaged number of controller interventions was computed

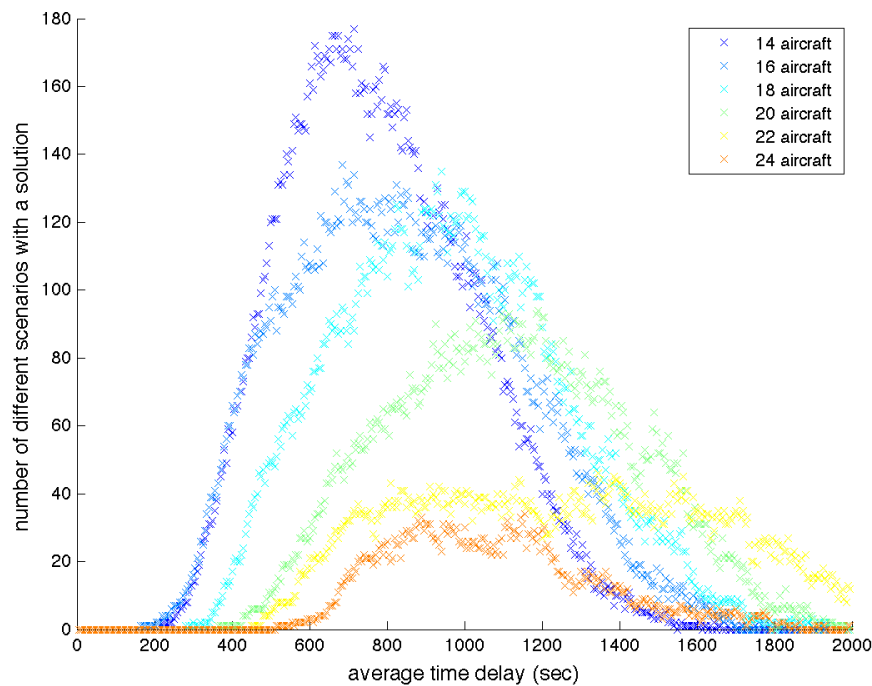


Figure 7: Distribution of scenarios having a solution in the delay range [0,2000] seconds

for each time delay bin. The standard deviation of the number of controller interventions was also computed for each time delay bin. Figure 8 and Figure 9 show the resulting curves.

3. Analysis

The set of graphs generated for this first experiment allows characterization of the solutions of the algorithm for different traffic densities. The scenario distribution in Figure 7 shows that the number of scenarios that have a solution in each simulation follows a distribution with a shape similar to the standard normal curve. However each simulation provides different distribution characteristics. As the total number of aircraft is increased, the number of scenarios having a solution decreases. The data analysis performed after each simulation demonstrates that the number of scenarios to be discarded from the solution set increases when the total number of aircraft is increased. A visual representation is provided in Figure 10.

The experiment results show that when the traffic density becomes greater than a total of 20 scheduled aircraft in a 30 minute time range, the algorithm is not able to compute solutions for each of the scenarios generated. For current day operations using the partial flows modeled in the Los Angeles terminal airspace, this does not present a problem because scenarios with more than 20 scheduled aircraft represent conditions 40% busier than typical. For example, when the total number of aircraft is set to 24, only 29.5% of the schedules generated have a solution. Therefore, for these larger values, the algorithm is stressed beyond its capacity to produce optimal solutions. Based on the data analysis of the results, one of the reasons for such behavior comes from the random computations of the number of arrivals and departures. In the scenario generation procedure, because the only requirement is on the total number of aircraft, extreme values of arrivals and departures are allowed to be generated. For examples, extreme values generated could be 1 arrival for 19 departures and vice versa. When running schedules with such values, the algorithm cannot delay aircraft more than the delay range defined in the code. In consequence, the algorithm cannot compute a solution for these schedules. Infeasible scenarios which distributions are represented in Figure 10 are thus characterized by schedules with extreme values of arrivals and departures.

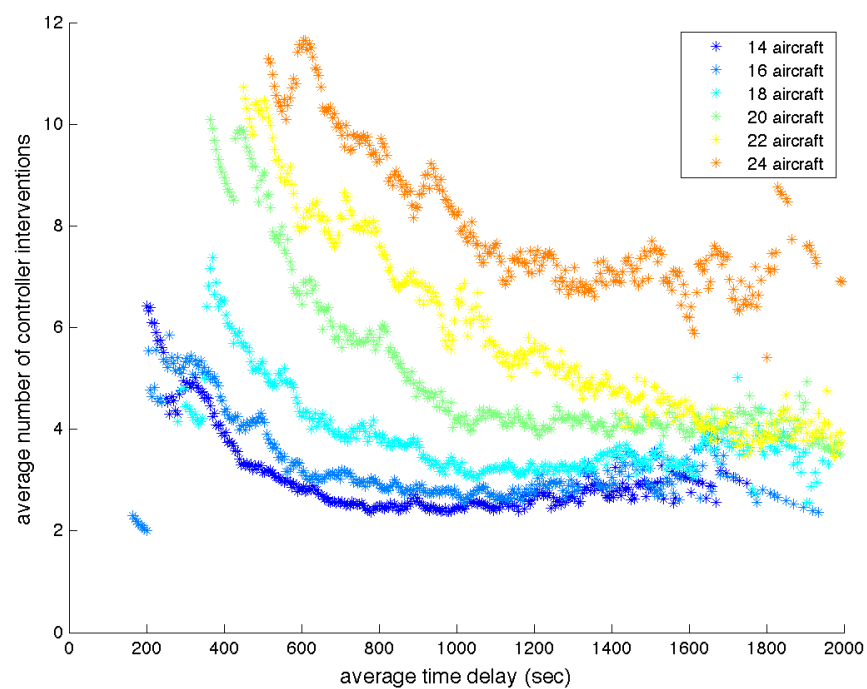


Figure 8: Average distribution of the number of controller interventions in the delay range [0,2000] seconds

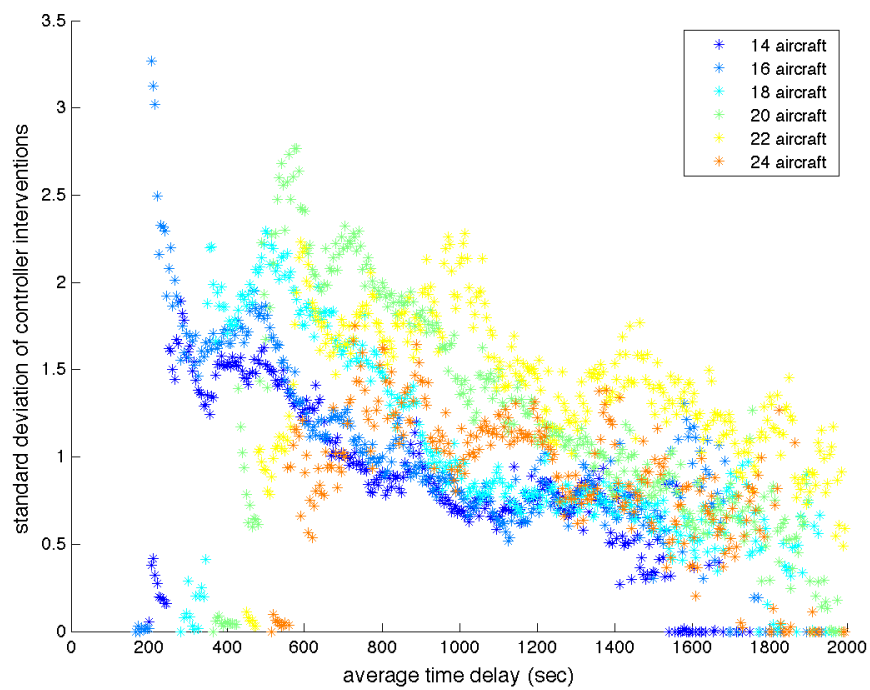


Figure 9: Standard deviation distribution of the number of controller interventions in the delay range [0,2000] seconds

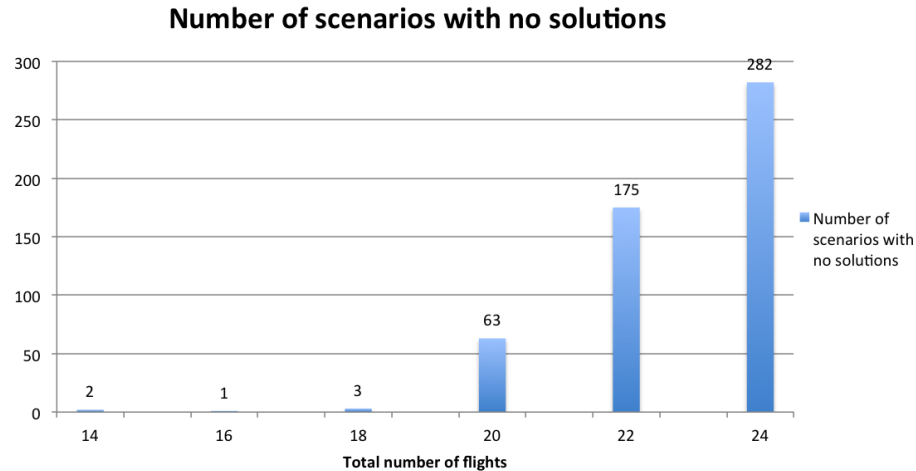


Figure 10: Number of scenarios to discard of the solution set

According to Figure 8 and Figure 9, one can observe that there exists a clear trade-off between delays and number of controller interventions. To obtain low delays, higher numbers of controller interventions are needed. This trend is even more noticeable as the total number of aircraft becomes larger. Figure 7 shows that each simulation has a range of delay for which the algorithm computes the most solutions. Therefore, highest confidence should be given to data corresponding to these ranges in Figure 8. Data points corresponding to average time delay of 200 seconds when the number of aircraft is equal to 16 are not representative of the main trend because the number of scenarios corresponding is low. For each time delay bin, the number of controller interventions increases as the total number of aircraft is increased. Figure 9 provides the statistical dispersion of the data from the average values presented in Figure 8. Observations of the standard deviation graph demonstrates that most of the data points obtained are concentrated and the number of outliers is low. For each simulation, when focusing on the most represented time delay range, the variability of the averaged number of controller interventions is low and confined. In this specific time delay range and for each simulation, the deviation of the average is no more than two controller interventions. Therefore, the computed solution is not deteriorated too much when testing different schedule-scenarios and this is indicative of algorithm robustness.

B. Experiment 2: Variation of the arrival-to-departure ratio

1. Setup

The second experiment focuses on varying the arrival-to-departure ratio. Because results from experiment one demonstrate that when the number of aircraft is greater than 20, the algorithm cannot compute a solution for each scenario tested, the second experiment only considers aircraft number between 14 and 20. Four different cases are investigated where each case corresponds to a fixed number of aircraft. In each case, the arrival-to-departure ratio is varied such that the only constraint is the number of aircraft. The number of arrivals is computed randomly between 1 and the fixed total number of aircraft and the number of departures is determined by the difference between the total number of aircraft and the number of arrivals. For each value of the number of aircraft, 400 schedule-scenarios are generated then tested in different simulations. The arrival-to-departure ratios ranged between 0.05 and 8.00. Because in real operations and for the partial flows considered, the arrival-to-departure ratio is typically 3, it is covered in the range of values tested.

2. Results

Figures 11-12-13 present the results of the four different cases. The same set of data from experiment 1 for a given fixed number of aircraft was segregated by arrival-to-departure ratio. Then a time discretization was performed on

the range of delay obtained. Finally, the number of scenarios having a solution in each time delay bin was computed. Figure 11 shows the scenario distributions of each case study when varying the arrival-to-departure ratio.

For each fixed number of aircraft case, the leading edge of the Pareto fronts with the same arrival-to-departure ratio was retrieved and saved. When every simulation was done, the global leading edge was computed from the ten different leading edges. The global delay range was sampled with a time delay bin size of five seconds, and the average number of controller interventions was computed for each time delay bin. The standard deviation of the number of controller interventions was also computed for each time delay bin. Figure 12 and Figure 13 show the resulting curves after identifying the different ratio values tested.

3. Analysis

Figure 11 provides the four different scenario distributions corresponding to different numbers of aircraft. The figures show that the number of scenarios having a solution in each of the 4 cases follow a distribution with similar properties to a standard normal curve. However, as the number of aircraft is increased the distributions are shifted to the right in time and they present fewer scenarios having a solution.

Two trends can be extracted from Figure 12. On one hand the ratio variation does not influence the results as much as the total number of aircraft. As the number of aircraft is increased, the Pareto fronts are shifted to the upper-right corner of the figures and the computed delay range is vertically translated to the right. Moreover, the number of controller interventions increases with the aircraft number considered in the scenarios. However when observing individually the different cases, it can be seen that larger ratios produce less number of controller interventions than smaller ratios for the whole range of time delay. Smaller ratios corresponds to scenarios with larger number of departures whereas larger ratios corresponds to scenarios with larger number of arrivals. Increasing the number of departures increases the number of controller interventions because departing aircraft are slower than arriving aircraft and because they require larger separation distances than arrivals. Wake vortex separations are imposed at the runway and larger amounts of uncertainty are modeled for departures than arrivals in this work.

According to Figure 13, a similar result trend can be observed for all case studies. As the number of aircraft considered in the scenarios is increased, the standard deviation of the number of controller interventions is shifted to the right in time. Moreover from this figure, high confidence can be given to these results. Most of the data points obtained have a low standard deviation and the number of outliers is low. Figure 13 shows that lower amount of data points can be observed for large ratios when the number of aircraft is 18 or 20. This confirms previous trends. Finally when focusing on the most represented time delay range, the variability of the averaged number of controller interventions is low and the deviation of the average does not exceed one. Therefore, this experiment is indicative of algorithm robustness.

C. Discussions

Because of the GPU introduction and accelerating strategies investigated, the significant computation time speedup allows the stochastic scheduler to be run hundreds of scenarios under reasonable time for fast-time simulations. The experiments that were designed enable a diverse range of scenarios to be tested and help to identify trade-offs of integrated departures and arrivals between reduced number of controller interventions and low delays. It is important to keep in mind that the schedules were generated randomly and therefore do not necessarily represent realistic conditions. However they covered parameter values corresponding to typical operations for the partial flows considered in the Los Angeles terminal airspace. The goal of the proposed methodology was to characterize the algorithm and its solution when computing very different sets of schedules. In reality, searching spaces of decision variables are often smaller in operations. Testing only real schedules obtained from historical data was not the focus of this study.

In this work, it was shown that when dense traffic conditions (number of aircraft greater than 20) are considered within a 30-minute time range, the algorithm is not often able to compute a solution because the algorithm cannot assign delays greater than the delay range defined in the code. However when applied to the Los Angeles terminal airspace, this trend corresponds to traffic conditions 40% greater than typical operations. Therefore this is not a problem for the Los Angeles application. The program was able to compute solutions for busy operations and show compromise solutions to help decision maker reduce controller interventions and achieve low delays. Moreover, it was found that varying the traffic density affects more significantly the results than varying the arrival-to-departure ratio. The total number of aircraft considered and the number of departures have the biggest impacts on the results obtained with this algorithm.

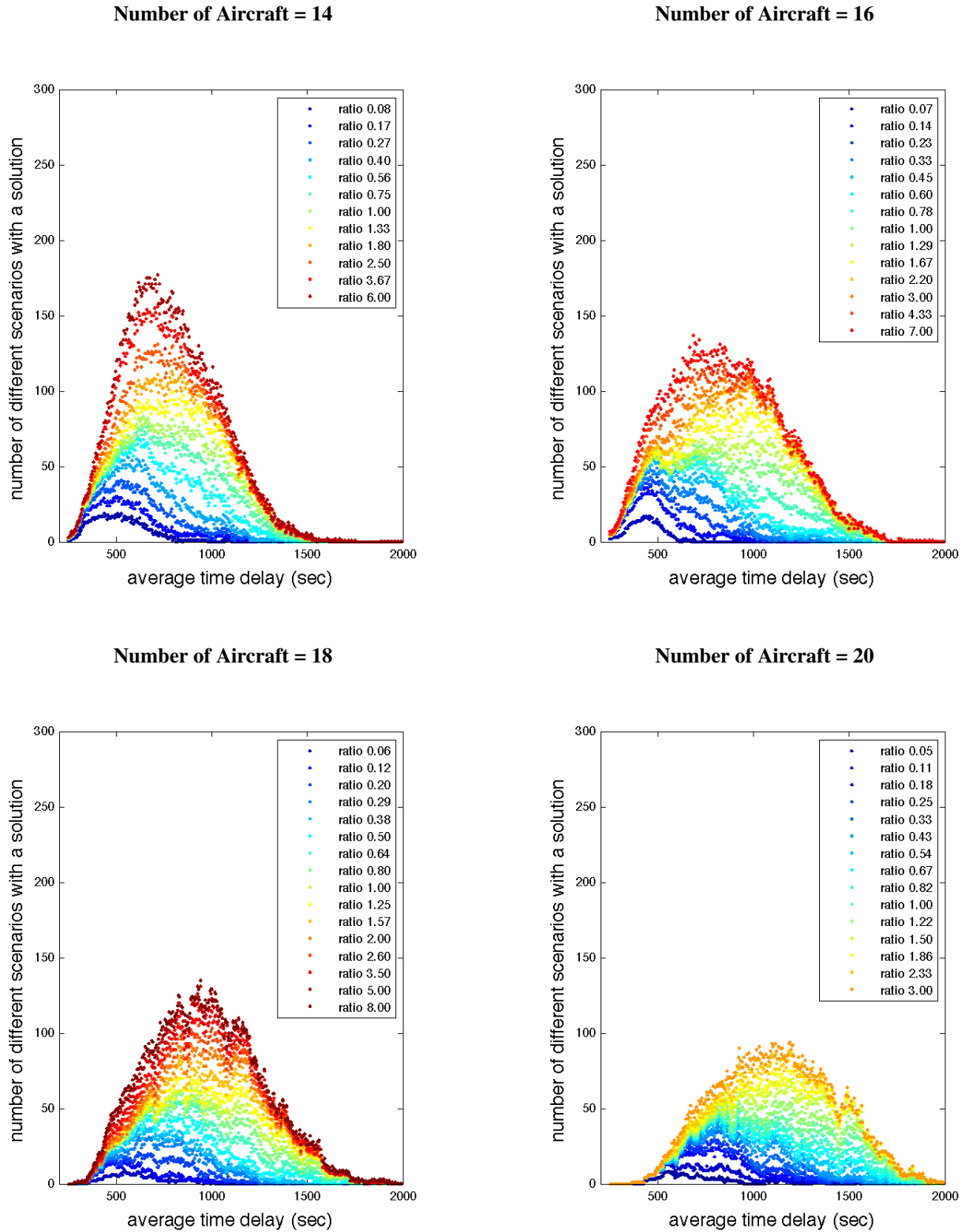


Figure 11: Distribution of scenarios having a solution in the delay range [0,2000] seconds

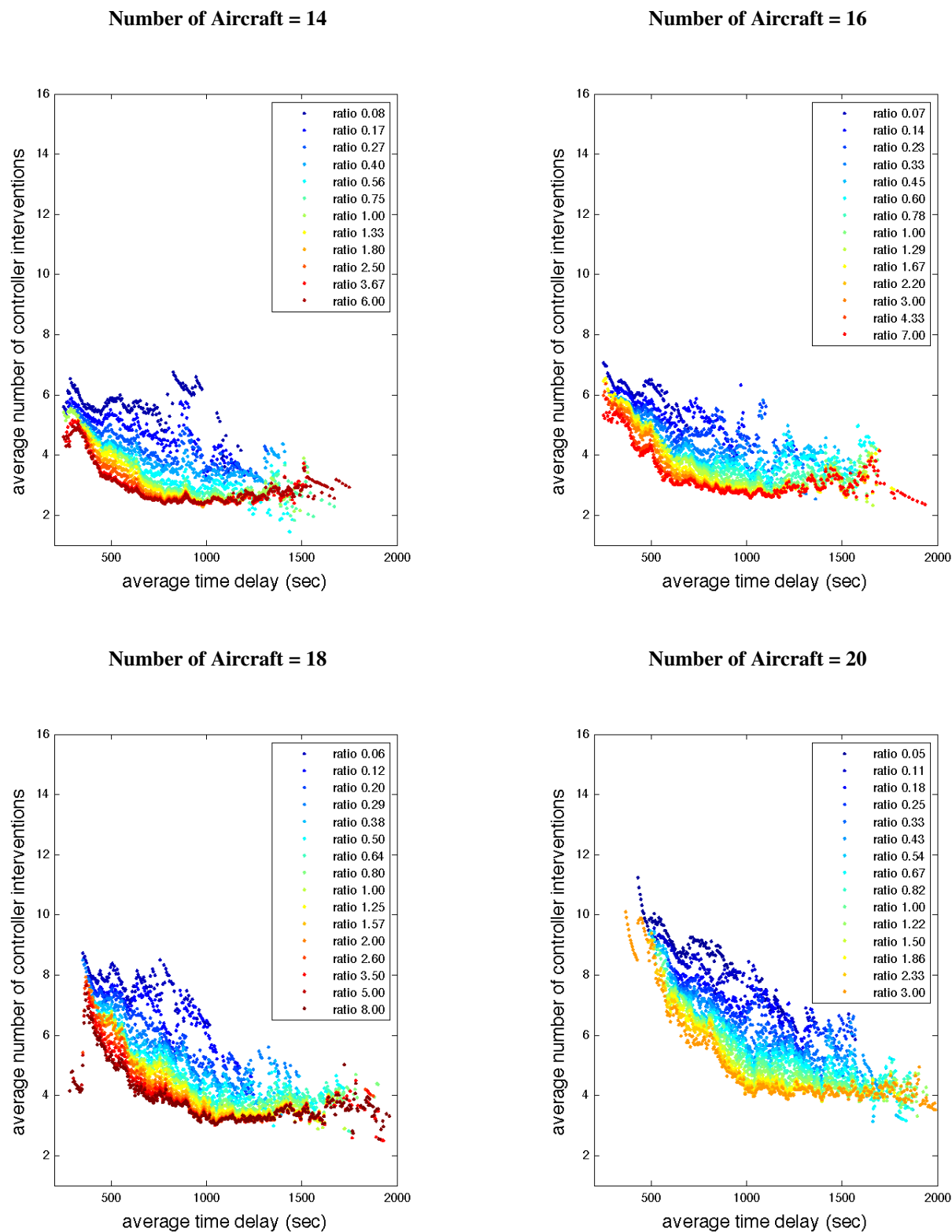


Figure 12: Average distribution of the number of controller interventions in the delay range [0,2000] seconds

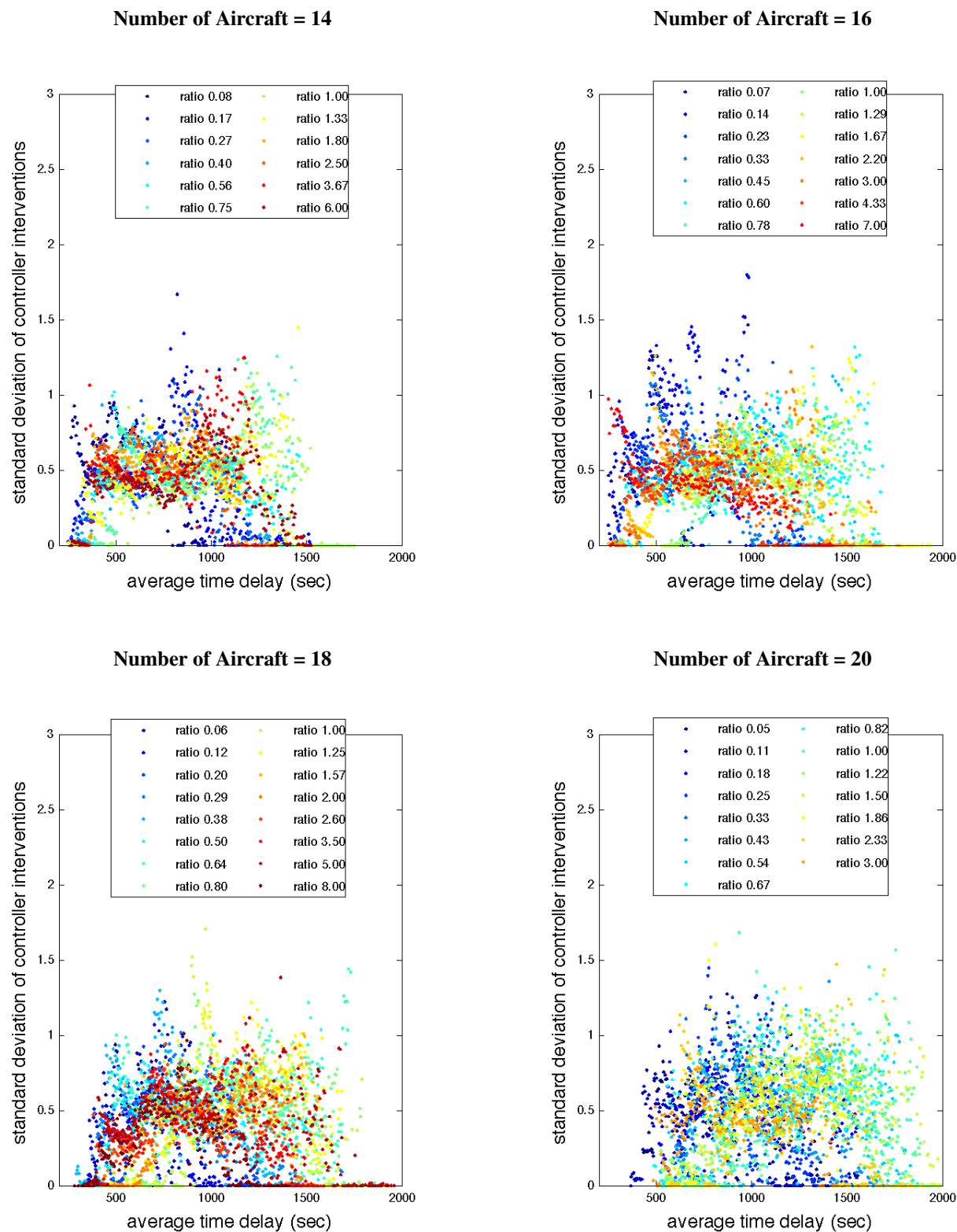


Figure 13: Standard deviation distribution of the number of controller interventions in the delay range [0,2000] seconds

The random nature of the scenario generation allowed the production of extreme valued schedules in both flight times and total number of aircraft. In the experiments, the algorithm was pushed to its limits when computing such scenarios. Within the thousands of scenarios tested, a range of schedules can be identified for which the algorithm computes realistic solutions. It would require additional work to further analyze their properties.

Although the acceleration methods proposed in this work were applied to stochastic scheduling, speedup gain and insights from this study can be used for other applications requiring real time application. Example of applications are search algorithms (e.g. genetic/evolutionary algorithms) and machine learning (e.g. neural networks).

VI. Conclusion

A GPU implementation of Monte Carlo simulations was developed using a mixed CPU-GPU development environment to obtain a massive computation parallelization. The GPU implementation of the stochastic scheduler is a success. Transferring the Monte Carlo computations to the GPU enabled a speedup of 637x. When the entire algorithm is parallelized on the mixed CPU-GPU development environment, the computation time is about 2.5 minutes to run a representative traffic flow scenario for partial flows in the Los Angeles terminal airspace. This computation time represents an acceleration of 154x the original run time. Despite relatively important CPU overhead times, the GPU is still able to accelerate the entire process and the speedup performance enabled to solve fast an air traffic management problem. To support a traffic variation analysis, several scheduling experiments were generated and tested on a model of the Los Angeles terminal airspace covering partial flows. The new computation time enabled testing different traffic configurations, characterize the computed solutions and elucidate the behavior of the algorithm itself. Although the scenarios were generated randomly and some of them did not represent realistic conditions, the scenarios that were tested produced realistic amount of delays and number of controller interventions. When comparing the generated scenarios with typical operations on the partial flows modeled in the Los Angeles terminal airspace, the program was able to compute solutions for traffic conditions up to 40% more than typical operations. There are two main take-aways from the results. First, there exists a clear compromise between achieving low time delays and low number of controller interventions. Moreover, the exploration of a variety of schedule-scenarios and the low dispersion of the results proves great algorithm robustness. Second, it was found that the variation of traffic density and the number of departures have greater impacts on the results than the variation of arrival-to-departure ratios. Moreover, it was found that for scenarios having extreme valued flight times or extreme number of aircraft, the algorithm could not compute a solution because of the fixed authorized delay range in the code. To complete the traffic variation analysis, future work should be conducted to test scenarios generated from historical data.

This paper proves that massive parallel implementation on Graphics Processing Units enables fast/real time solution to air traffic management applications. The integrated arrival/departure scheduling problem with uncertainty can be solved in realistic times when using GPUs. This approach outperforms the traditional serial implementation on CPUs developed in previous work despite the use of multi-threading on serial processors. This paper encourages the use of such techniques to remove research struggles associated with air traffic management numerical optimization techniques.

References

- ¹Dear, R. G., "The Dynamic Scheduling of Aircraft in the Near Terminal Area," Tech. rep., Cambridge, Mass.: Flight Transportation Laboratory, Massachusetts Institute of Technology, 1976.
- ²Dear, R. G. and Sherif, Y. S., "An algorithm for Computer Assisted Sequencing and Scheduling of Terminal Area Operations," *Transportation Research Part A: General*, Vol. 25, No. 2, 1991, pp. 129–139.
- ³Neuman, F. and Erzberger, H., *Analysis of Delay Reducing and Fuel Saving Sequencing and Spacing Algorithms for Arrival Traffic*, Citeseer, 1991.
- ⁴Balakrishnan, H. and Chandran, B., "Scheduling Aircraft Landings under Constrained Position Shifting," *AIAA Guidance, Navigation, and Control Conference*, Keystone, CO, 2006.
- ⁵Beasley, J. E., Krishnamoorthy, M., Sharaiha, Y. M., and Abramson, D., "Scheduling Aircraft Landings The Static Case," *Transportation science*, Vol. 34, No. 2, 2000, pp. 180–197.
- ⁶Kupfer, M., "Scheduling Aircraft Landings to Closely Spaced Parallel Runways," *Eighth USA/Europe Air Traffic Management Research and Development Seminar (ATM2009)*, Napa, CA, June 2009, pp. 1–10.
- ⁷Gupta, G., Malik, W., and Jung, Y. C., "A Mixed Integer Linear Program for Airport Departure Scheduling," *9th AIAA Aviation Technology, Integration, and Operations Conference (ATIO)*, Hilton Head, South Carolina, 2009.

- ⁸Atkin, J. A. D., Burke, E. K., Greenwood, J. S., and Reeson, D., "A Metaheuristic Approach to Aircraft Departure Scheduling at London Heathrow Airport," Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 235–252.
- ⁹Rathinam, S., Wood, Z., Sridhar, B., and Jung, Y. C., "A generalized dynamic programming approach for a departure scheduling problem," *AIAA Guidance, Navigation, and Control Conference*, Chicago, IL, 2009.
- ¹⁰Chandran, B. and Balakrishnan, H., "A Dynamic Programming Algorithm for Robust Runway Scheduling," *American Control Conference*, 2007. ACC '07, New York, NY, 2007, pp. 1161–1166.
- ¹¹Sölveling, G., *Stochastic Programming Methods for Scheduling of Airport Runway Operations under Uncertainty*, Ph.D. thesis, Georgia Institute of Technology, Aug. 2012.
- ¹²Xue, M. and Zelinski, S., "Optimal Integration of Departures and Arrivals in Terminal Airspace," *AIAA Guidance, Navigation, and Control Conference*, Minneapolis, MN, August 2012.
- ¹³Xue, M. and Zelinski, S., "Optimization of Integrated Departures and Arrivals Under Uncertainty," *AIAA 13th Aviation Technology, Integration, and Operation Conference (ATIO)*, Los Angeles, CA, August 2013, pp. 1–9.
- ¹⁴Xue, M., Zelinski, S., and Mulfinger, D., "Uncertainty Study of Integrated Departures and Arrivals: A Los Angeles Case Study," *AIAA 13th Aviation Technology, Integration, and Operation Conference (ATIO)*, Los Angeles, CA, August 2013, pp. 1–10.
- ¹⁵Chen, H., Zhao, Y. J., and Provan, C., "Multiple-Point Integrated Scheduling of Terminal Area Traffic," *Journal of Aircraft*, Vol. 48, No. 5, 2011, pp. 1646–1657.
- ¹⁶Chen, H., Zhao, Y. J., and Provan, C., "Dynamic Real-time Scheduling of Terminal Traffic," *AIAA Guidance, Navigation, and Control Conference*, Portland, Oregon, August 2011.
- ¹⁷Capozzi, B. and Atkins, S., "A Hybrid Optimization Approach to Air Traffic Management for Metroplex Operations," *10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, Forth Worth, TX, 2010.
- ¹⁸Capozzi, B., Atkins, S., and Choi, S., "Towards Optimal Routing and Scheduling of Metroplex Operations," *9th AIAA Aviation Technology, Integration, and Operations Conference (ATIO)*, Hilton Head, South Carolina, 2009.
- ¹⁹DeLaurentis, D. A. and Ayyalasomayajula, S., "Analysis of Dependencies and Impacts of Metroplex Operations," Tech. rep., 2010.
- ²⁰Saraf, A., Clarke, J.-P., and McClain, E., "Discussion and Comparison of Metroplex-Wide Arrival Scheduling Algorithms," *10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, Forth Worth, TX, 2010.
- ²¹Atkins, S., "Observation and Measurement of Metroplex Phenomena," *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, 2008, p. 3.
- ²²Li, L., Park, J. W., and Clarke, J.-P., "A Simulation-based Method for Estimating Metroplex Efficiency," *Digital Avionics Systems Conference (DASC), 2011 IEEE/AIAA 30th*, 2011, p. 2.
- ²³Thippavong, J. and Mulfinger, D., "Design Considerations for a New Terminal Area Arrival Scheduler," *10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, Forth Worth, TX, September 2010.
- ²⁴Zone, N. D., "CUDA Toolkit Documentation," .
- ²⁵Zone, N. D., "CURAND :: CUDA Toolkit Documentation," .
- ²⁶Che, S., Li, J., Sheaffer, J. W., Skadron, K., and Lach, J., "Accelerating Compute-Intensive Applications with GPUs and FPGAs," *2008 Symposium on Application Specific Processors (SASP)*, IEEE, 2008, pp. 101–107.
- ²⁷Matsuoka, S., Aoki, T., Endo, T., Nukada, A., Kato, T., and Hasegawa, A., "GPU accelerated computing—from hype to mainstream, the rebirth of vector computing," *Journal of Physics: Conference Series*, Vol. 180, August 2009, pp. 012043.
- ²⁸Adriaens, J. T., Compton, K., Kim, N. S., and Schulte, M. J., "The case for GPGPU spatial multitasking," *2012 IEEE 18th International Symposium on High Performance Computer Architecture (HPCA)*, 2012, pp. 1–12.
- ²⁹Joldes, G. R., Wittek, A., and Miller, K., "Real-time nonlinear finite element computations on GPU—Application to neurosurgical simulation," *Computer methods in applied mechanics and engineering*, 2010, pp. 3305–3314.
- ³⁰Stone, S., Haldar, J., Tsao, S., Hwu, W.-m., Liang, Z.-P., and Sutton, B., "Accelerating Advanced MRI Reconstructions on GPUs," *CF'08*, ACM Press, Ischia, Italy, 2008, p. 251.
- ³¹Gaikwad, A. and Toke, I. M., "GPU based Sparse Grid Technique for Solving Multidimensional Options Pricing PDEs," *the 2nd Workshop*, ACM Press, Portland, Oregon, 2009, pp. 1–9.
- ³²Gaikwad, A. and Toke, I. M., "Parallel Iterative Linear Solvers on GPU: A Financial Engineering Case," *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2010.
- ³³Tandale, M. D., Wiraatmadja, S., Vaddi, V., and Rios, J. L., "Massively Parallel Optimal Solution to the Nationwide Traffic Flow Management Problem," *AIAA 13th Aviation Technology, Integration, and Operation Conference (ATIO)*, Los Angeles, CA, August 2013.
- ³⁴Oiso, M., Matsumura, Y., Yasuda, T., and Ohkura, K., "Implementing Genetic Algorithms to CUDA Environment Using Data Parallelization," *Tehnički vjesnik 18, 4(2011)*, 511–517, July 2013, pp. 1–7.
- ³⁵Shah, R., Narayanan, P., and Kothapalli, K., "GPU-Accelerated Genetic Algorithms," *Proceedings of The 3rd Workshop on Parallel Architectures for Bio-inspired Algorithms (WPABA) in conjunction with Parallel Architectures for Compilation Techniques (PACT'10)*, Vienna, Austria, 2010.
- ³⁶Pospíchal, P. and Jaros, J., "Gpu-based acceleration of the genetic algorithm," *GECCO'09 competition*, 2009.
- ³⁷Li, J., Lv, X., and Liu, L., "A Parallel Genetic Algorithm with GPU Accelerated for Large-scale MDVRP in Emergency Logistics," *2011 IEEE 14th International Conference on Computational Science and Engineering (CSE)*, 2011, pp. 602–605.
- ³⁸Wahib, M., Munawar, A., Munetomo, M., and Akama, K., "Optimization of Parallel Genetic Algorithms for NVIDIA GPUs," *2011 IEEE Congress on Evolutionary Computation (CEC)*, 2011, pp. 803–811.
- ³⁹Harding, S. and Banzhaf, W., "Fast Genetic Programming on GPUs," *Genetic Programming*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 90–101.
- ⁴⁰Timar, S. D., Nagle, G., Saraf, A., Yu, P., Hunt, P., Trapani, A., and Johnson, N., "Super Density Operations Airspace Modeling for the Southern California Metroplex," *AIAA Modeling and Simulation Technology Conference*, 2011.

⁴¹Sridhar, B. and Chen, N., “Short-Term National Airspace System Delay Prediction Using Weather Impacted Traffic Index,” *Journal of guidance, control, and dynamics*, Vol. 32, No. 2, 2009, pp. 657–662.

⁴²Engelland, S. A. and Capps, A., “Trajectory-Based Takeoff Time Predictions Applied to Tactical Departure Scheduling: Concept Description, System Design, and Initial Observations,” *AIAA Aviation Technology, Integration, and Operations Conference (ATIO)*, Virginia Beach, VA, 2011.

⁴³“Box-Muller transform — Wikipedia, The Free Encyclopedia,” 2014.

⁴⁴NVIDIA Corporation, *CUDA Warps and Occupancy*, Sept. 2011.

⁴⁵Legriel, J., Le Guernic, C., Cotton, S., and Maler, O., “Approximating the Pareto Front of Multi-Criteria Optimization Problems,” January 2010, pp. 1–15.

⁴⁶Ngatchou, P., Zarei, A., and El-Sharkawi, M., “Pareto multi objective optimization,” *Proceedings of the 13th International Conference on Intelligent Systems Application to Power Systems, 2005.*, IEEE, 2005, pp. 84–91.